

Pipeline Depth Tradeoffs and the Intel® Pentium® 4 Processor

**Doug Carmean
Principal Architect
Intel Architecture Group**

August 21, 2001

Agenda

- **Review**
- **Pipeline Depth**
- **Execution Trace Cache**
- **L1 Data Cache**
- **Summary**

Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel® products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This document contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product is available. Verify with your local sales office that you have the latest datasheet before finalizing a design.

Intel processors may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel, Pentium, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and foreign countries.

Copyright © (2001) Intel Corporation.

Intel® Netburst™ Micro-architecture vs P6

Basic P6 Pipeline

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sc	Wb	Exec

Intro at
733MHz
180nm

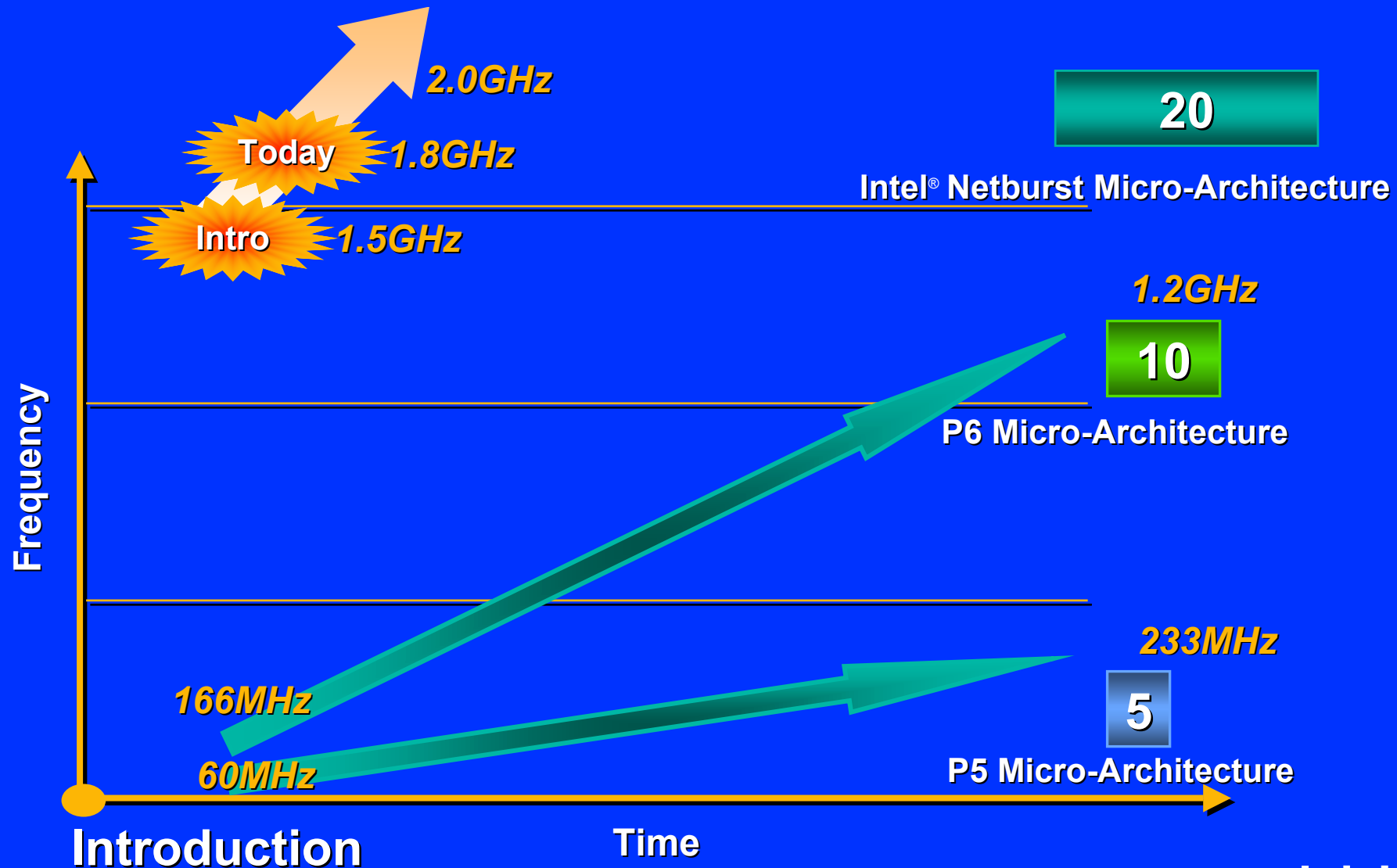
Basic Pentium® 4 Processor Pipeline

Intro at

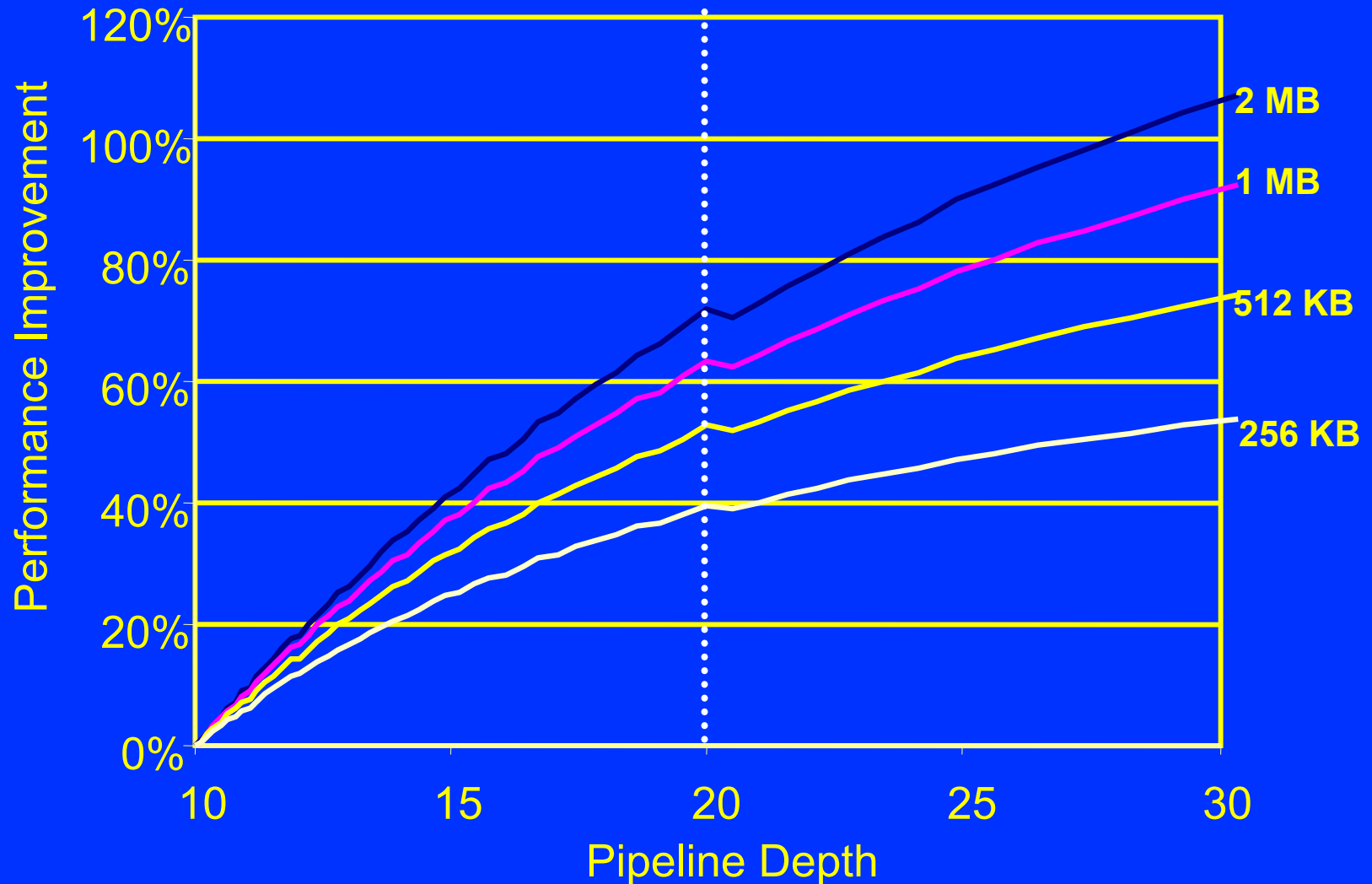
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
TC	Nxt IP	TC	Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	Wb	Wb	Wb

Hyper pipelined Technology enables industry leading performance and clock rate

Hyper Pipelined Technology



Deeper Pipelines are Better



Why not deeper pipelines?

- **Increases complexity**
 - **Harder to balance**
 - **More challenges to architect around**
 - **More algorithms**
 - **Greater validation effort**
 - **Need to pipeline the wires**

**Overall Engineering Effort Increases Quickly
as Pipeline depth increases**

Performance

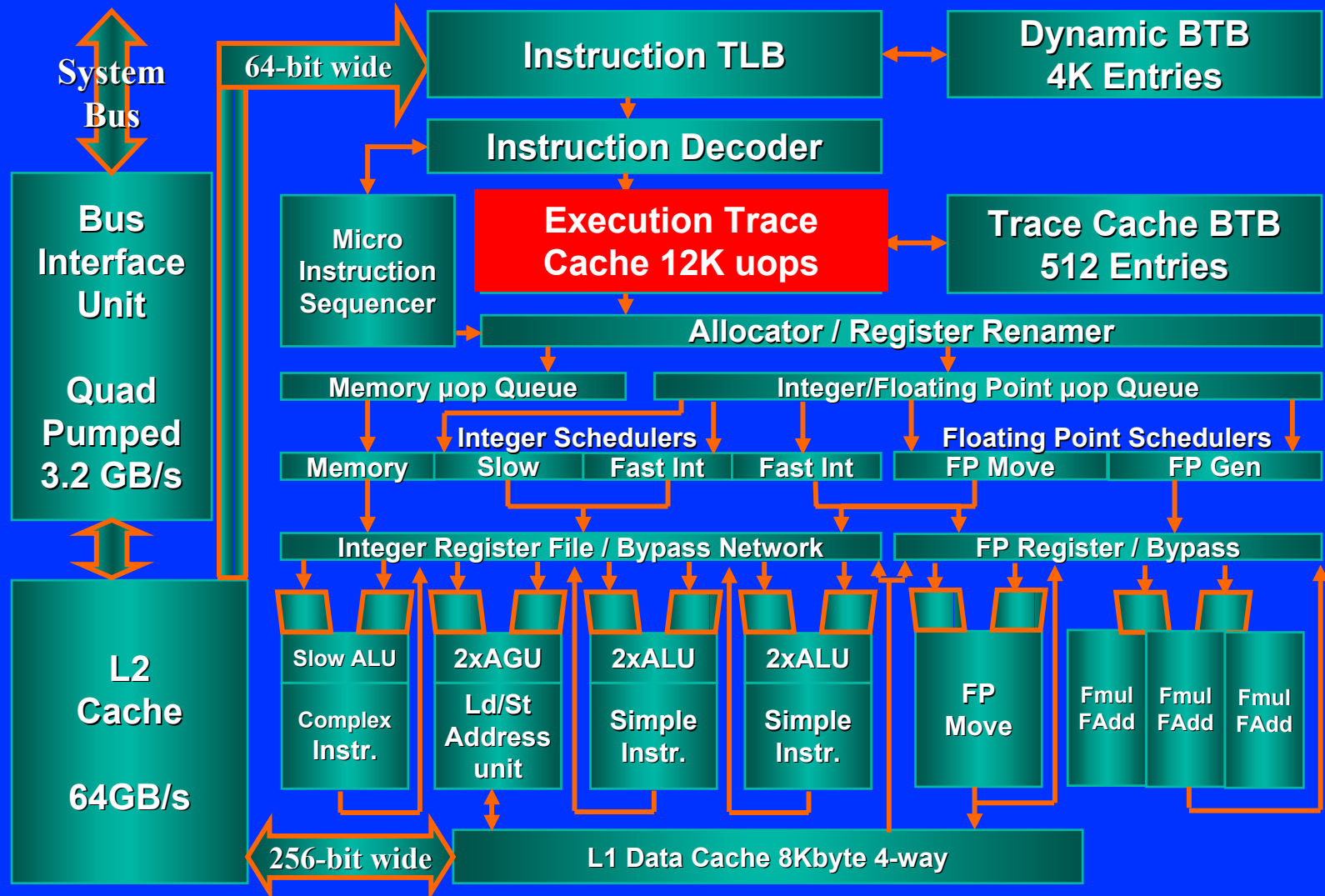
- High Bandwidth Front End
- High Bandwidth Front End

High Bandwidth Front End

Higher Frequency increases requirements of front end

- **Branch prediction is more important**
 - So we improved it
- **Need greater uop bandwidth**
 - Branches constantly change the flow
 - Need to decode more instructions in parallel

Block Diagram



Execution Trace Cache

1 cmp
2 br -> T1

... (unused code)

T1: 3 sub

4 br -> T2

... (unused code)

T2: 5 mov
6 sub

7 br -> T3

... (unused code)

T3: 8 add
9 sub

10 mul
11 cmp
12 br -> T4

Trace Cache Delivery

1	cmp	2	br T1	3	T1: sub
4	br T2	5	mov	6	sub
7	br T3	8	T3: add	9	sub
10	mul	11	cmp	12	br T4

Execution Trace Cache

P6 Microarchitecture

1	cmp	2	br	T1
3	T1: sub	4	br	T2
5	mov	6	sub	7 br T3
8	T3:add	9	sub	10 mul
11	cmp	12	br	T4

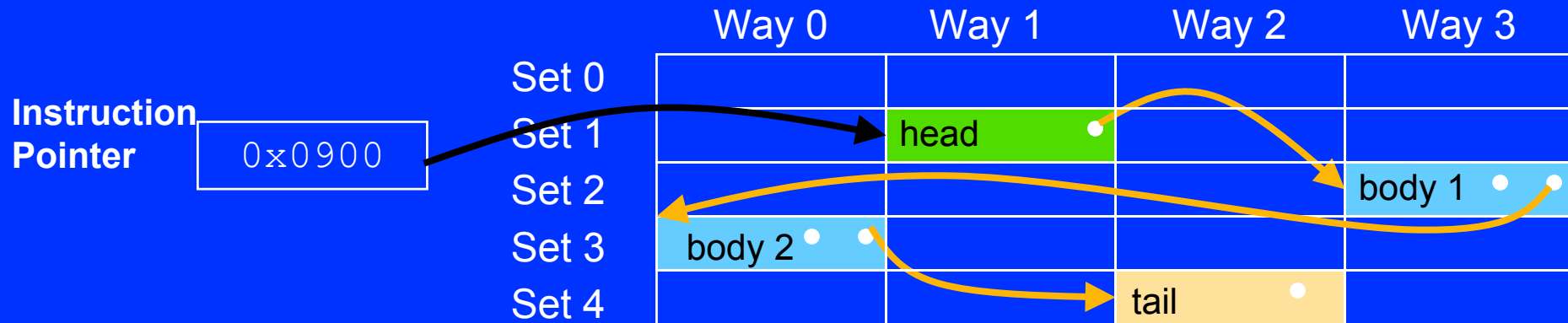
BW = 1.5 uops/ns

Trace Cache Delivery

1	cmp	2	br	T1	3	T1: sub
4	br	T2	5	mov	6	sub
7	br	T3	8	T3:add	9	sub
10	mul	11	cmp	12	br	T4

BW = 6 uops/ns

Inside the Execution Trace Cache



head

cmp, br T1, T1:sub, br T2, mov, sub

body 1

br T3, T3:add, sub, mul, cmp, br T4

body 2

T4:add, sub, mov, add, add, mov

tail

add, sub, mov, add, add, mov

Self Modifying Code

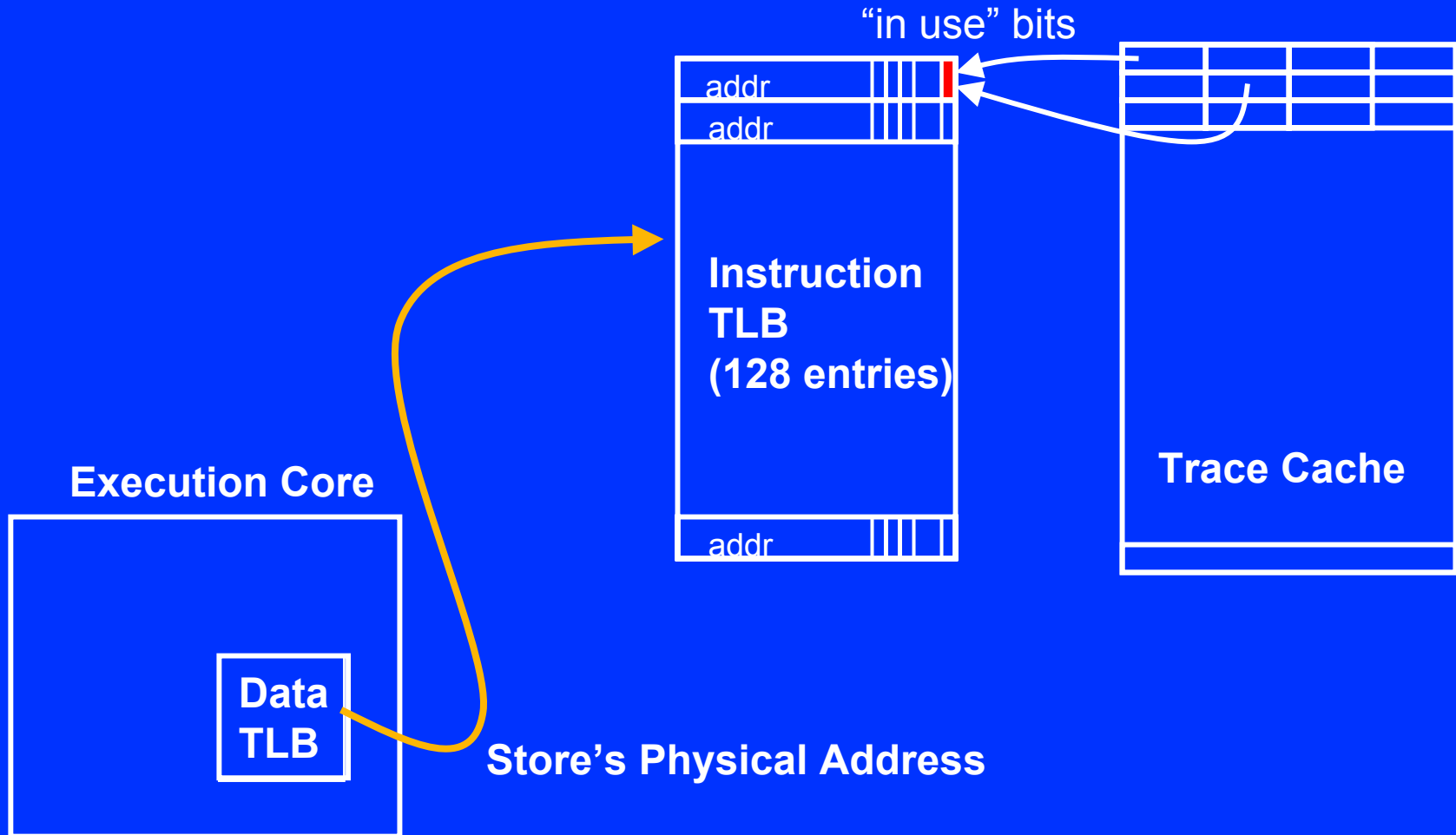
- **Programs that modify the instruction stream that is being executed**
- **Very common in Java* code from JITs**
- **Requires hardware mechanisms to maintain consistency**

*Other names and brands may be claimed as the property of others.

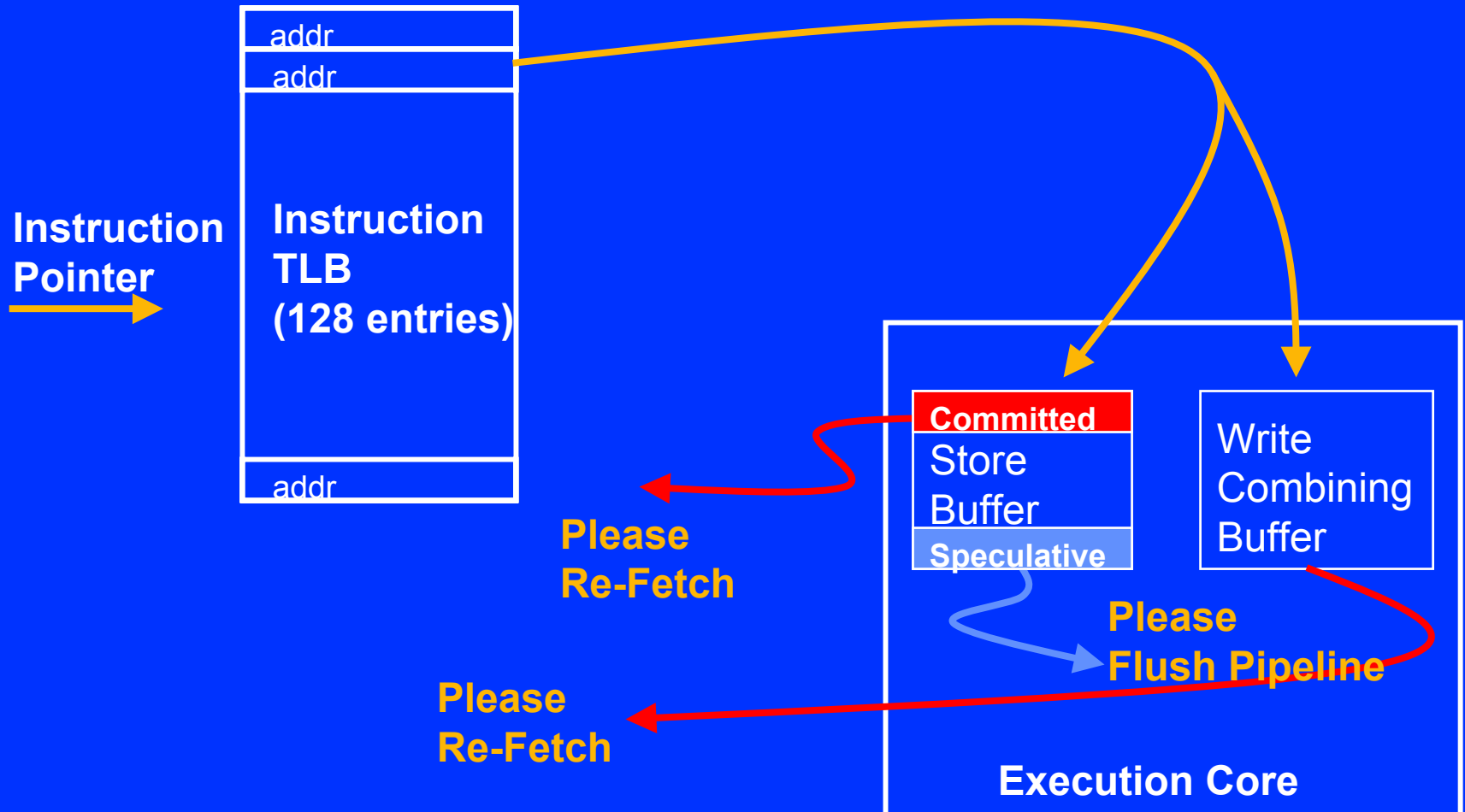
Self Modifying Code

- **The hardware needs to handle two basic cases:**
 - Stores that write to instructions in the Trace Cache
 - Instruction fetches that hit pending stores
 - Speculative
 - Committed

Case 1: Stores to cached instructions



Case 2: Fetches to pending stores



Execution Trace Cache

- Provides higher bandwidth for higher frequency core
- Reduces fetch latency
- Requires new fundamentally new algorithms

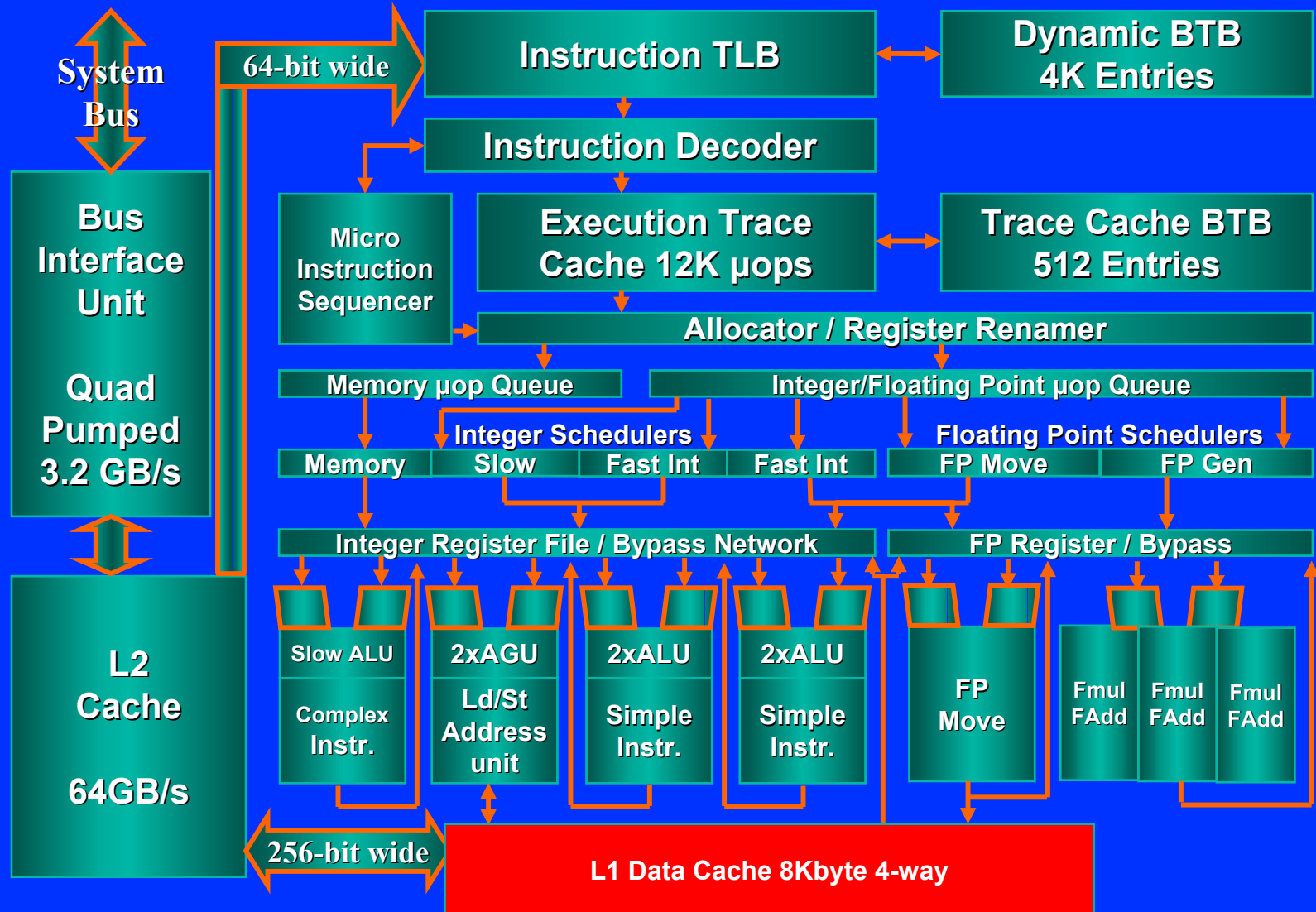
Performance

- High bandwidth front end
- Low



Low Latency Core

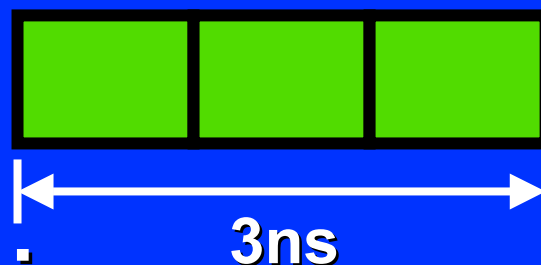
L1 Data Cache



L1 Cache is 3x Faster

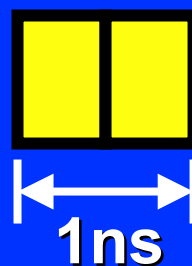
- P6:

- 3 clocks @ 1GHz



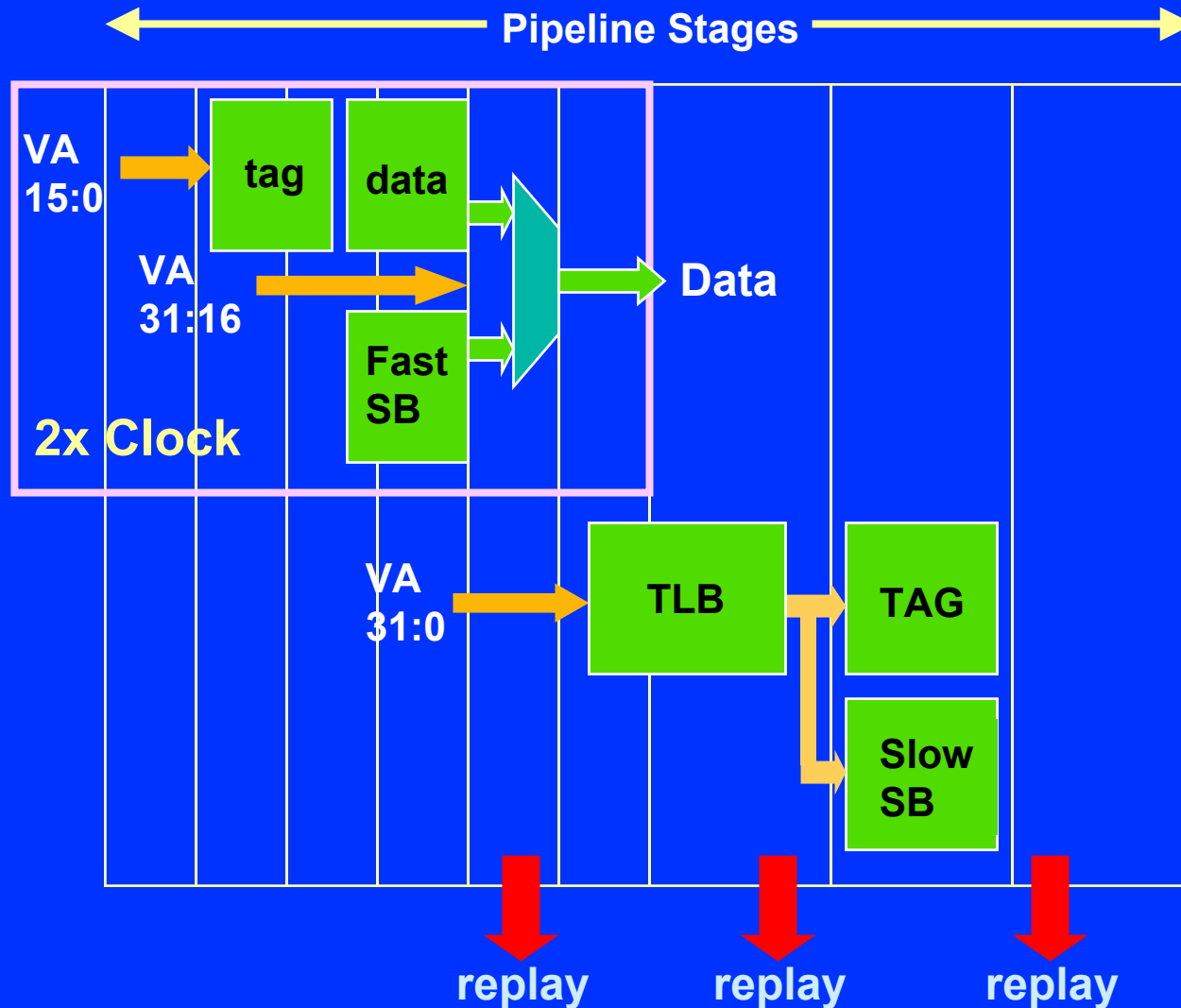
- Pentium® 4 Processor :

- 2 clocks @ 2GHz

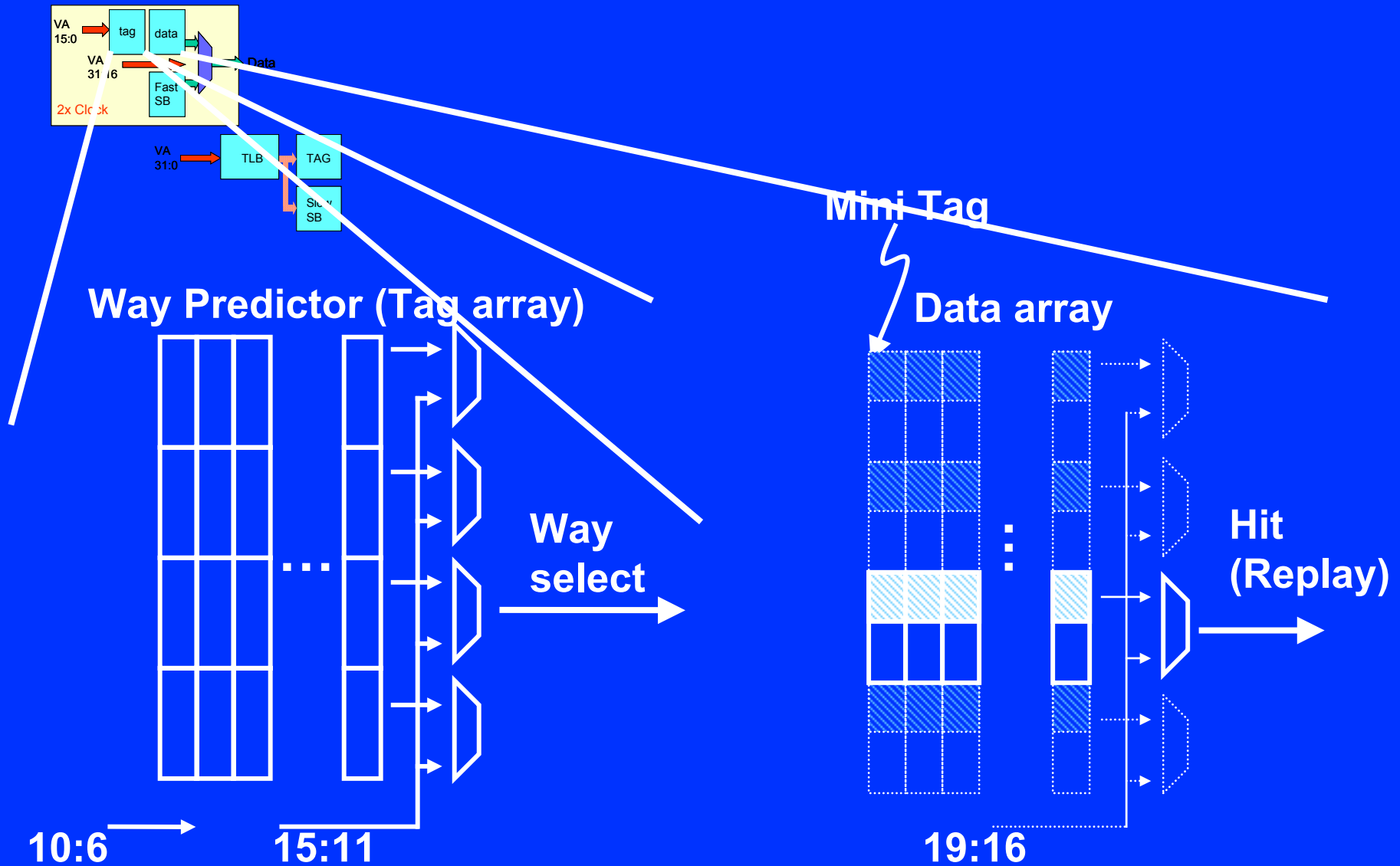


Lower Latency is Higher Performance

L1 Data Cache



L1 Data Cache



Performance

- High bandwidth front end
- Low latency core
- Low

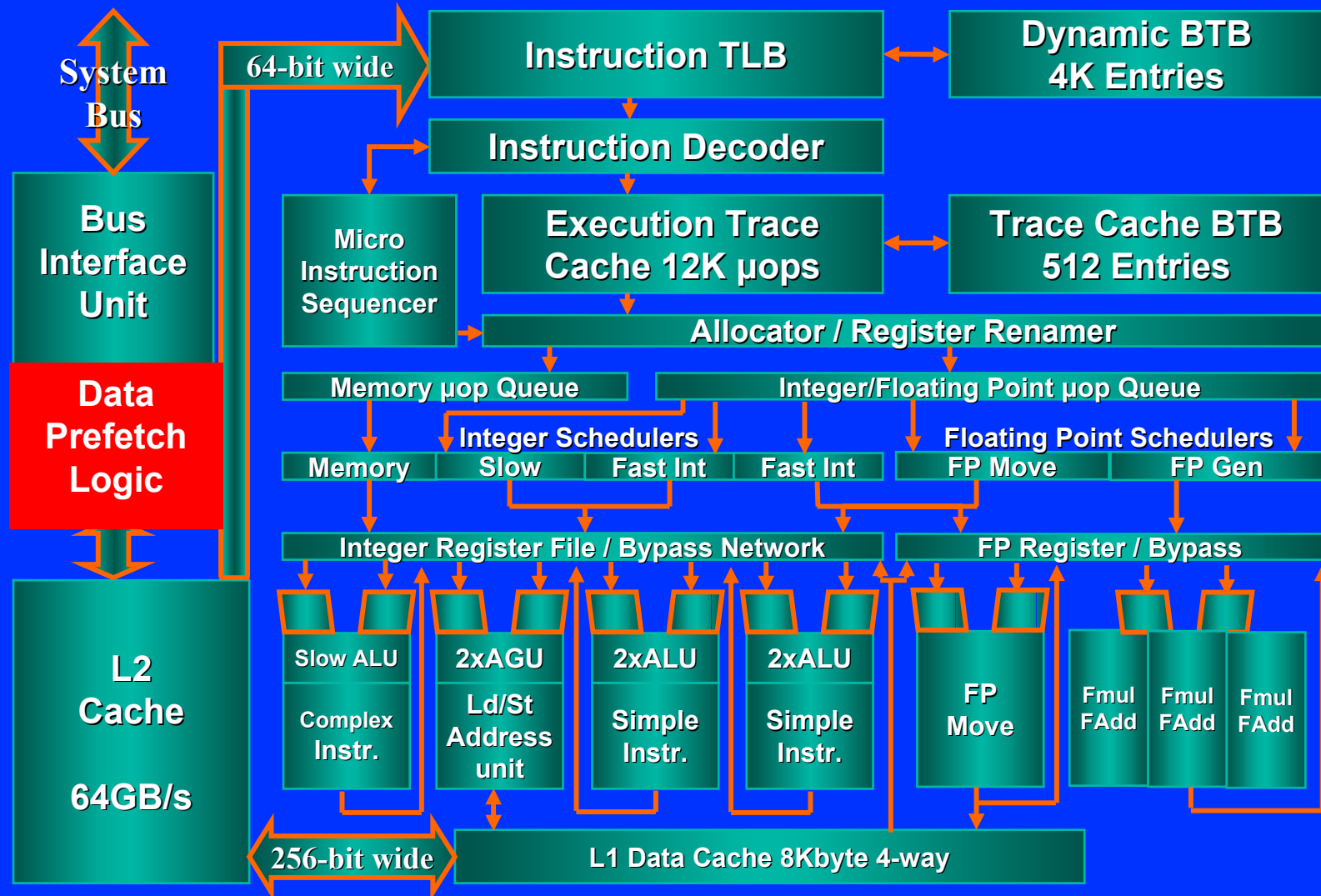


Lower Memory Latency

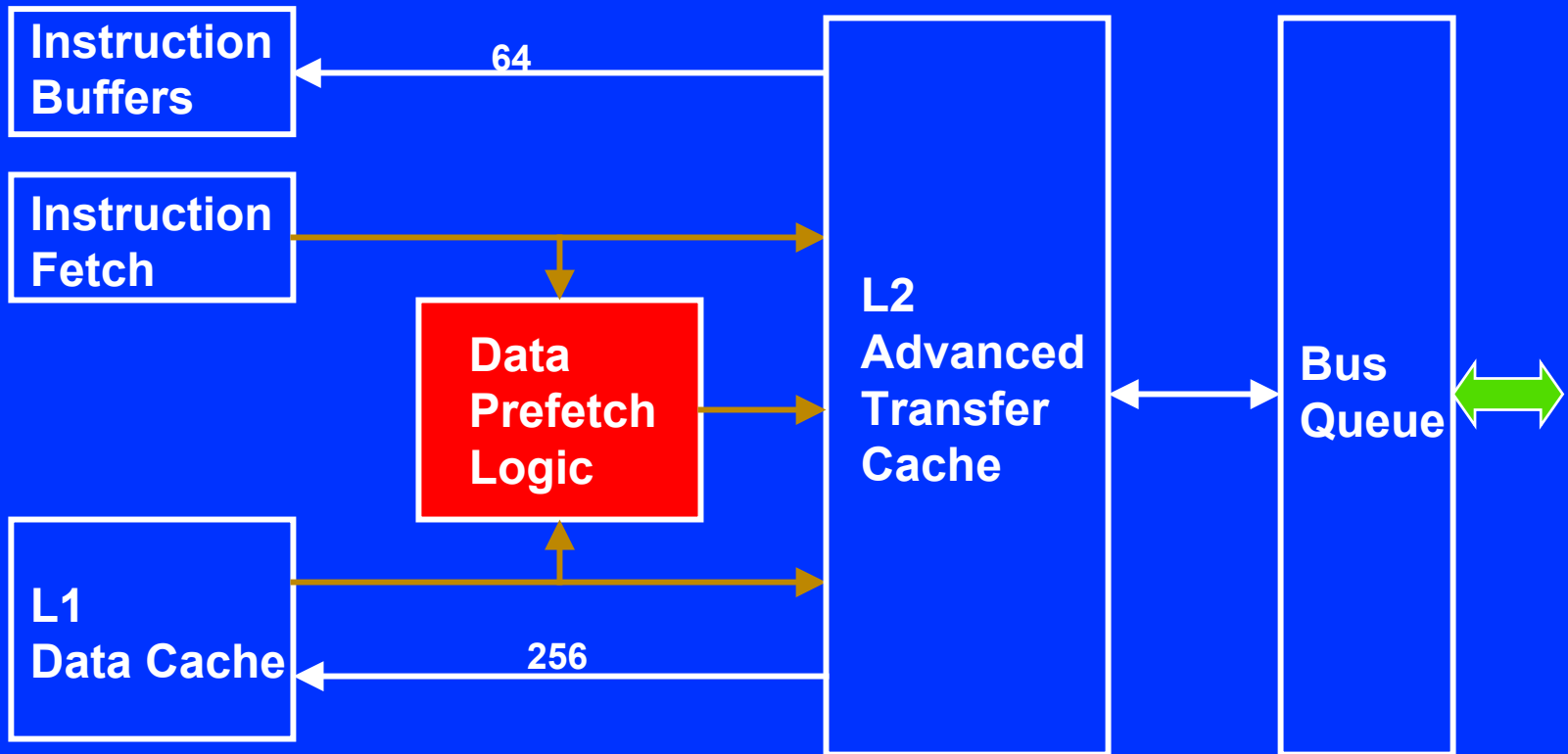
Reducing Latency

- **As frequency increases, it is important to improve the performance of the memory subsystem**
- **Data Prefetch Logic**
 - **Watches processor memory traffic**
 - **Looks for patterns**
 - **Initiates accesses**

Data Prefetch Logic



Data Prefetch Logic



Prefetch logic first checks L2 cache and then fetches lines from memory that miss L2 cache.

Data Prefetch Logic

- **Watches for streaming memory access patterns**
 - Can track 8 multiple independent streams
 - Loads, Stores or Instruction
 - Forward or Backward
- **Analysis on 32 byte cache line granularity**
- **Looks for “mostly” complete streams:**
 - Access to cache lines 1,2,3,4,5,6 will prefetch
 - Access to cache lines 1,2, 4,5,6 will prefetch
 - 1, ,3, , ,6, , ,9 will not prefetch

Performance

- **High bandwidth front end**
- **Low latency core**
- **Lower memory latency**

Summary

- **The Pentium® 4 Processor's deep pipelines provide high performance by enabling high frequency**
- **Deep pipelines are more difficult to engineer**
- **Larger caches further improve benefits of Pentium® 4 Processor's deep pipeline**
- **Compilers further increase benefits of deep pipelines by removing hazards**

